(22F) CST8277 Enterprise Application Programming

Assignment 3: ACME College (JPA Mapping & JUnit Tests)

Please read this document carefully. If your submission does not meet the requirements as stated here, you may lose marks even though your program runs. Additionally, this assignment is also a teaching opportunity, that is **materials presented in this document may be on the midterm and/or the final exam!**

Overview

For Assignment 3, ACME Corp. has tasked you to map their college database scheme to POJOs. Also, ACME Corp. would like you to write JUnit tests to confirm that the JPA mappings you added are correct. Finally, you are required to produce a Maven 'Surefire' pass-fail report for all of your test cases.

Theme for Assignment 3

After completing this assignment, you will have achieved the following:

1. Gain experience in mapping a real-world database scheme to POJOs
2. Gain experience in creating complex JPA queries
3. Gain experience in building JUnit test cases
4. Use the Maven 'Surefire' plugin to generate a pass-fail report for all test cases

Download and Importing the Project to Your Eclipse Workspace

1. Download *ACMECollege-JPA-Mapping-JUnit-Skeleton.zip*
2. Unzip the folder content to some location
3. In Eclipse, go to **File/Import…/Existing Maven Projects**
4. Navigate to your project folder
5. Make sure that there is a check mark beside the project's pom.xml file
6. Click **Finish**
7. To generate the <Classname>_.java files, right-click on the project name, select **Properties**
8. Select **JPA** on the left, and on **Canonical metamodel (JPA 2.0)**, on the **Source folder** dropdown, select src/main/java, and finally click **Apply and Close**; after you fix the errors on the classes, the <Classname>_.java files will be auto-generated

After Importing the Project to Your Eclipse Workspace

Do the following steps to view all the TODOs/tasks and hints for this assignment:

1. Go to **Window** menu, select **Preferences**
2. Type "task" on the "type filter text" textbook on the left
3. Under **Java Compiler**, you should see "Task Tags"
4. Select **Task Tags** and on the right, and then click **New…**
5. On the **Tag** text box, type "Hint"
6. On the **Priority** dropdown, select **Low** and click **OK**
7. Click **Apply** (Note: You have to let Eclipse rebuild your project if it asks.) and then click **Apply and Close**
8. If you don't see the **Tasks** on your Eclipse, do the following:

   a. Go to **Window** menu, select **Show View**, select **Other…**
   b. Under **General**, select **Tasks** and click **Open**
9. To filter the tasks, do the following:
   a. On the **Tasks** bar, select the filter icon on the right
   b. It should bring up the **Filters** window
   c. Make sure that **Show all items** is not checked
   d. Make sure that **TODOs** is checked
   e. Under **Scope**, make sure that **On elements in select project** is selected
   f. Then click **New…**, select **New Configuration**, click **Rename**, enter "Hints" on the **Configuration Name** pop-up, and then click **OK**
   g. Again, make sure that **On elements in selected project** is selected for Hints
   h. Then under **Description**, type in "Hint" beside **Text: contains** and click **Apply and Close**
   i. Then make sure that **Show all items** checkbox is checked on the **Filters** window

Approach for This Assignment

1. First of all, finish all the TODOs in the models (e.g. Student, Course, MembershipCard, ClubMembership, StudentClub, AcademicStudentClub, NonAcademicStudentClub, etc.); you may use the Professor and CourseRegistration entities as guides/references in completing the other entities/models
2. Refer to the attached ER diagram (*ACMECollege-Diagram.pdf*) to understand how the entities and tables are related to each other; you must annotate the classes/entities based on how they are related as shown in *ACMECollege-Diagram.pdf*
3. After finishing the TODOs in the models correctly, you should be able to run *ACMECollegeDriver.java* as a Java application successfully
4. Also, after finishing the TODOs in the models correctly, next is to write your test cases for the rest of the models (a sample has already been provided for CourseRegistration (i.e. *TestCRUDCourseRegistrationOriginal.java*)); **Note:** You should be able to run *TestCRUDCourseRegistrationOriginal.java* successfully after fixing the models.
5. At the very least, you have to write test cases for Student, Professor, Course, & MembershipCard
6. Note that you have to also finish the TODOs in *JUnitBase.java*; the methods getTotalCount(), getAll(), getWithId(), and getCountWithId() are all optional and you do not have to complete them…however, they will greatly help reduce code repetition once you have completed them and used them in your own JUnit test classes

Note: The *ACMECollegeDriver.java* is the driver class for the project. It is a basic class and you do not have to do anything here. When it is executed, it initializes the database for the project. It also populates the database with some data as can be seen inside the addSampleData() method. The driver class uses the following files to initialize the database:

- *src/main/resources/META-INF/persistence.xml*
- *src/main/resources/META-INF/sql/acmecollege-drop.sql*
- *src/main/resources/META-INF/sql/acmecollege-create.sql*

Inside *persistence.xml* file, it defines the following properties:

```
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>

<property name="javax.persistence.schema-generation.create-source" value="script"/>
<property name="javax.persistence.schema-generation.create-script-source" value="META-
INF/sql/acmecollege-create.sql"/>

<property name="javax.persistence.schema-generation.drop-source" value="script"/>
```

```
<property name="javax.persistence.schema-generation.drop-script-source" value="META-INF/sql/acmecollege-drop.sql"/>
```

As can be seen above, every time the application is run, it first drops the database (using the script *acmecollege-drop.sql*) and then creates it (using the script *acmecollege-create.sql*).

JUnit Tests

In this assignment, JUnitBase should serve as the base class of your JUnit test classes that you will create. The following methods are required to be implemented:

- deleteAllData()
- deleteAllFrom() – this method is called by deleteAllData() several times to delete the contents of tables in the database

The following methods below are not required to be implemented. However, completing and invoking them will greatly help reduce code repetition in completing this assignment.

- getTotalCount()
- getAll()
- getWithId()
- getCountWithId()

Once the above 4 methods are implemented, you can invoke them in your JUnit test classes.

Note: Method setupAll() (annotated with @BeforeAll) is invoked before all test cases are executed and method tearDownAll() (annotated with @AfterAll) is called after all test cases are performed.

Please refer to *TestCRUDCourseRegistrationOriginal.java* on how to write your JUnit test classes for the remaining entities.

To order your test cases, use the following annotations in your JUnit test class (please also see *TestCRUDCourseRegistrationOriginal.java*):

- `@TestMethodOrder(MethodOrderer.OrderAnnotation.class)`
- `@Order(1), @Order(2), @Order(3), …`

Ideally, each test case should be completely independent of other test cases.

Note that for each JUnit test class that you write, it should contain around **six (6)** individual test cases (similar to *TestCRUDCourseRegistrationOriginal.java*).

Build and Test Your Assignment

Once you have completed the tasks above, do the following:

1. Right-click on your *ACMECollegeDriver.java* and choose **Run As** and then choose **Java Application**
2. Make sure there are no errors reported on the console
3. Then, run your JUnit tests:
   a. To run your JUnit tests individually, right-click on your JUnit test class (e.g. *TestCRUDCourseRegistrationOriginal.java*) and choose **Run As** and then choose **JUnit Test**; make sure that all your test cases passed
   b. To build your project and run all the test cases, right-click on your project, choose **Run As**, select **Maven build…**, on the **Goals** text box type (without the double-quotes):

“**clean install test surefire-report:report site -DgenerateReports=false**”

c. The above will generate *target/site/surefire-report.html* file which will show the results of the test cases when opened with a web browser

Submitting Assignment 3

To submit your Assignment 3, first you must export your completed project as a zip file. Eclipse has an export function to create an external archive, i.e. a zip file, of your project. Make sure to not include the “target” subfolder when generating the zip file (see screenshots below). You should name your zip file as **<lastname>-<firstname>-Assignment3.zip**, for example, **Smith-John-Assignment3.zip**. You must then upload the correct generated zip file to Brightspace under Assignment 3 (Brightspace/Activities/Assignments/Assignment 3).

Please make sure that your completed project compiles and builds successfully, has good indentation and coding style, not overly complicated, and is efficient. The completed code must demonstrate your understanding of how JPA mappings, JPA queries, and JUnit tests work.

Note: Do not forget to upload the correct generated zip file to Brightspace under Assignment 3.

Demoing Assignment 3

For Assignment 3, please note that demo is not required. However, if you think you have completed your project early (before the due date), you can demo your Assignment 3 to your lab professor during your respective lab session before the due date. The advantage of doing an optional demo for your Assignment 3 is that if the demo is done before the due date, then you will have a chance to fix issues/bugs found during the demo before you submit your completed project on Brightspace. Please note that you are allowed to keep submitting updated versions of your project on Brightspace (before the due date) though only the latest version will be considered your official submission and it will be the one to be graded by your lab professor.

– end –